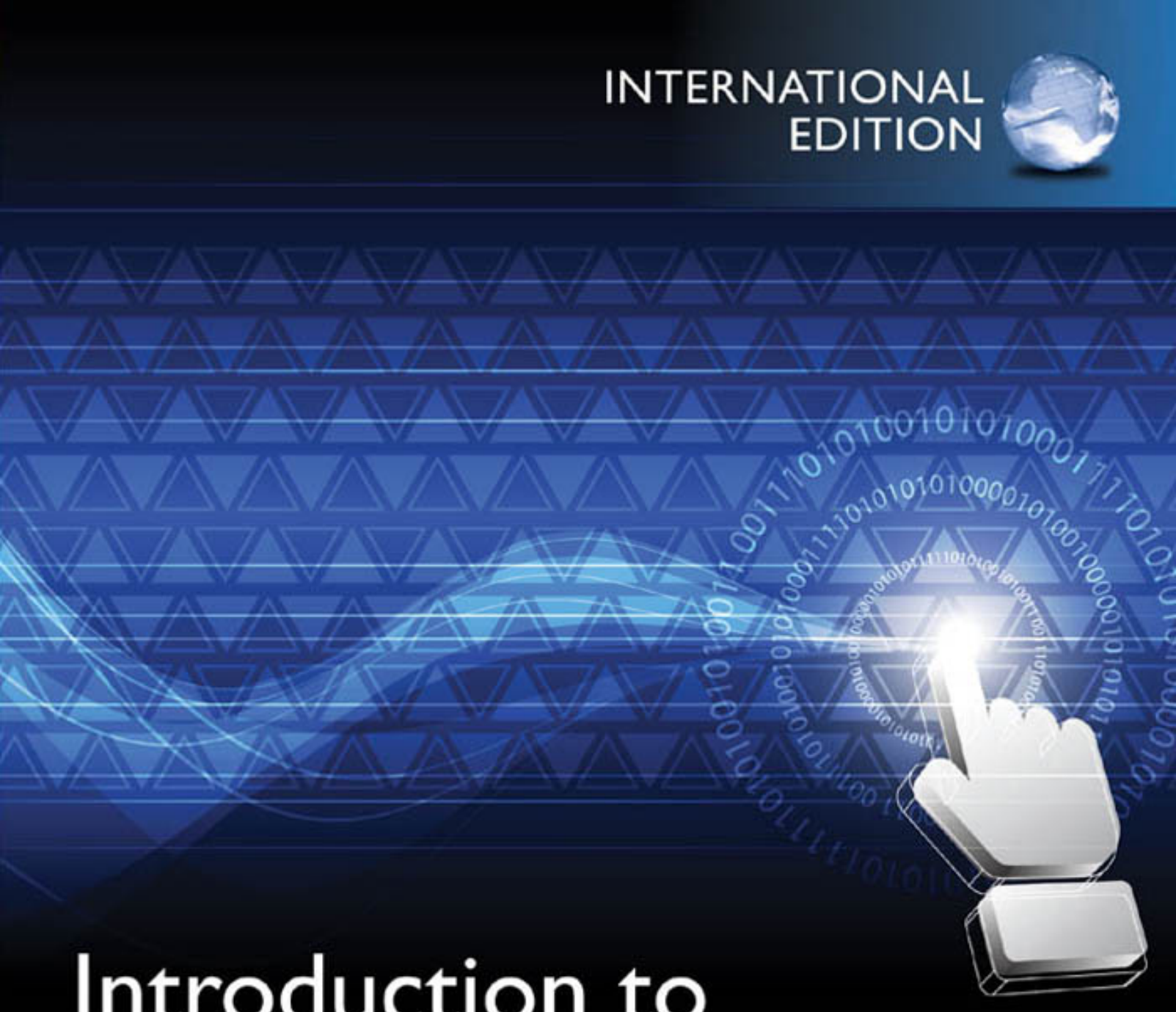
INTERNATIONAL EDITION



# Introduction to Programming with C++

THIRD EDITION

Y. Daniel Liang

ALWAYS LEARNING

PEARSON

# ONLINE ACCESS

Thank you for purchasing a new copy of ***Introduction to Programming with C++,*** **Third Edition**. Your textbook includes six months of prepaid access to the book's Companion Website. This prepaid subscription provides you with full access to the following student support areas:

- VideoNotes
- Web Chapters
- Student supplements

**Use a coin to scratch off the coating and reveal your student access code.**
**Do not use a knife or other sharp object as it may damage the code.**

To access the *Introduction to Programming with C++, Third Edition*, Companion Website for the first time, you will need to register online using a computer with an Internet connection and a web browser. The process takes just a couple of minutes and only needs to be completed once.

1. Go to **www.pearsoninternationaleditions.com/liang**
2. Click on **Companion Website**.
3. Click on the **Register** button.
4. On the registration page, enter your student access code* found beneath the scratch-off panel. Do not type the dashes. You can use lower- or uppercase.
5. Follow the on-screen instructions. If you need help at any time during the online registration process, simply click the **Need Help?** icon.
6. Once your personal Login Name and Password are confirmed, you can begin using the *Introduction to Programming with C++* Companion Website!

### To log in after you have registered:

You only need to register for this Companion Website once. After that, you can log in any time at **www.pearsoninternationaleditions.com/liang** by providing your Login Name and Password when prompted.

*Important: The access code can only be used once. This subscription is valid for six months upon activation and is not transferable. If this access code has already been revealed, it may no longer be valid.

To implement this idea, use two variables, say `low` and `high`, to denote the position of two characters at the beginning and the end in a string `s`, as shown in Listing 5.16 (lines 13, 16). Initially, `low` is `0` and `high` is `s.length() - 1`. If the two characters at these positions match, increment `low` by `1` and decrement `high` by `1` (lines 27–28). This process continues until (`low >= high`) or a mismatch is found.

## LISTING 5.16 TestPalindrome.cpp

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7    // Prompt the user to enter a string
8    cout << "Enter a string: ";
9    string s;
10   getline(cin, s);                                    input string
11
12   // The index of the first character in the string
13   int low = 0;
14
15   // The index of the last character in the string
16   int high = s.length() - 1;
17
18   bool isPalindrome = true;
19   while (low < high)
20   {
21     if (s[low] != s[high])                            compare two characters
22     {
23       isPalindrome = false; // Not a palindrome       not palindrome
24       break;                                          exit loop
25     }
26
27     low++;
28     high--;
29   }
30
31   if (isPalindrome)
32     cout << s << " is a palindrome" << endl;
33   else
34     cout << s << " is not a palindrome" << endl;
35
36   return 0;
37 }
```

```
Enter a string: abccba ↵ Enter
abccba is a palindrome
```

```
Enter a string: abca ↵ Enter
abca is not a palindrome
```

The program declares a string (line 9), reads a string from the console (line 10), and checks whether the string is a palindrome (lines 13–29).

The `bool` variable `isPalindrome` is initially set to `true` (line 18). When comparing two corresponding characters from both ends of the string, `isPalindrome` is set to `false` if the

two characters differ (line 23). In this case, the `break` statement is used to exit the `while` loop (line 24).

If the loop terminates when `low >= high`, `isPalindrome` is true, which indicates that the string is a palindrome.

## 5.11 Case Study: Displaying Prime Numbers


*This section presents a program that displays the first 50 prime numbers in 5 lines, each containing 10 numbers.*

An integer greater than `1` is *prime* if its only positive divisor is `1` or itself. For example, `2`, `3`, `5`, and `7` are prime numbers, but `4`, `6`, `8`, and `9` are not.

The program can be broken into the following tasks:

- Determine whether a given number is prime.
- For `number` = `2`, `3`, `4`, `5`, `6`, . . . , test whether it is prime.
- Count the prime numbers.
- Display each prime number, and display ten numbers per line.

Obviously, you need to write a loop and repeatedly test whether a new `number` is prime. If the `number` is prime, increase the count by `1`. The `count` is `0` initially. When it reaches `50`, the loop terminates.

Here is the algorithm:

```
Set the number of prime numbers to be printed as
  a constant NUMBER_OF_PRIMES;
Use count to track the number of prime numbers and
  set an initial count to 0;
Set an initial number to 2;

while (count < NUMBER_OF_PRIMES)
{
  Test whether number is prime;

  if number is prime
  {
    Display the prime number and increase the count;
  }

  Increment number by 1;
}
```

To test whether a number is prime, check whether it is divisible by `2`, `3`, `4`, up to `number/2`. If a divisor is found, the number is not a prime. The algorithm can be described as follows:

```
Use a bool variable isPrime to denote whether
  the number is prime; Set isPrime to true initially;

for (int divisor = 2; divisor <= number / 2; divisor++)
{
  if (number % divisor == 0)
  {
    Set isPrime to false
    Exit the loop;
  }
}
```

The complete program is given in Listing 5.17.

## LISTING 5.17 PrimeNumber.cpp

```
1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4
5  int main()
6  {
7    const int NUMBER_OF_PRIMES = 50; // Number of primes to display
8    const int NUMBER_OF_PRIMES_PER_LINE = 10; // Display 10 per line
9    int count = 0; // Count the number of prime numbers                    count prime numbers
10   int number = 2; // A number to be tested for primeness
11
12   cout << "The first 50 prime numbers are \n";
13
14   // Repeatedly find prime numbers
15   while (count < NUMBER_OF_PRIMES)
16   {
17     // Assume the number is prime
18     bool isPrime = true; // Is the current number prime?                 check primeness
19
20     // Test if number is prime
21     for (int divisor = 2; divisor <= number / 2; divisor++)
22     {
23       if (number % divisor == 0)
24       {
25         // If true, the number is not prime
26         isPrime = false; // Set isPrime to false
27         break; // Exit the for loop                                      exit loop
28       }
29     }
30
31     // Display the prime number and increase the count
32     if (isPrime)                                                        display if prime
33     {
34       count++; // Increase the count
35
36       if (count % NUMBER_OF_PRIMES_PER_LINE == 0)
37         // Display the number and advance to the new line
38         cout << setw(4) << number << endl;
39       else
40         cout << setw(4) << number;
41     }
42
43     // Check if the next number is prime
44     number++;
45   }
46
47   return 0;
48 }
```

```
The first 50 prime numbers are
   2   3   5   7  11  13  17  19  23  29
  31  37  41  43  47  53  59  61  67  71
  73  79  83  89  97 101 103 107 109 113
 127 131 137 139 149 151 157 163 167 173
 179 181 191 193 197 199 211 223 227 229
```

subproblem

This is a complex program for novice programmers. The key to developing a programmatic solution for this problem, and for many other problems, is to break it into subproblems and develop solutions for each of them in turn. Do not attempt to develop a complete solution in the first trial. Instead, begin by writing the code to determine whether a given number is prime, then expand the program to test whether other numbers are prime in a loop.

To determine whether a number is prime, check whether it is divisible by a number between 2 and number/2 inclusive. If so, it is not a prime number; otherwise, it is a prime number. For a prime number, display it. If the count is divisible by 10, advance to a new line. The program ends when the count reaches 50.

The program uses the break statement in line 27 to exit the for loop as soon as the number is found to be a nonprime. You can rewrite the loop (lines 21–29) without using the break statement, as follows:

```
for (int divisor = 2; divisor <= number / 2 && isPrime;
     divisor++)
{
  // If true, the number is not prime
  if (number % divisor == 0)
  {
    // Set isPrime to false, if the number is not prime
    isPrime = false;
  }
}
```

However, using the break statement makes the program simpler and easier to read in this case.

## Key Terms

break statement 205
continue statement 206
do-while loop 188
for loop 191
infinite loop 178
input redirection 186
iteration 176
loop 176
loop body 176
loop-continuation-condition 176
nested loop 196
off-by-one error 178
output redirection 186
posttest loop 194
pretest loop 194
sentinel value 184
while loop 176

## Chapter Summary

1. There are three types of repetition statements: the while loop, the do-while loop, and the for loop.

2. The part of the loop that contains the statements to be repeated is called the *loop body*.

3. A one-time execution of a loop body is referred to as an *iteration of the loop*.

4. An *infinite loop* is a loop statement that executes infinitely.

5. In designing loops, you need to consider both the *loop control structure* and the loop body.

6. The `while` loop checks the `loop-continuation-condition` first. If the condition is `true`, the loop body is executed; if it is `false`, the loop terminates.

7. The `do-while` loop is similar to the `while` loop, except that the `do-while` loop executes the loop body first and then checks the `loop-continuation-condition` to decide whether to continue or to terminate.

8. The `while` loop and the `do-while` loop often are used when the number of repetitions is not predetermined.

9. A *sentinel value* is a special value that signifies the end of the loop.

10. The `for` loop generally is used to execute a loop body a fixed number of times.

11. The `for` loop control has three parts. The first part is an initial action that often initializes a control variable. The second part, the loop-continuation-condition, determines whether the loop body is to be executed. The third part is executed after each iteration and is often used to adjust the control variable. Usually, the loop control variables are initialized and changed in the control structure.

12. The `while` loop and `for` loop are called *pretest loops* because the continuation condition is checked before the loop body is executed.

13. The `do-while` loop is called a *posttest loop* because the condition is checked after the loop body is executed.

14. Two keywords, `break` and `continue`, can be used in a loop.

15. The `break` keyword immediately ends the innermost loop, which contains the break.

16. The `continue` keyword only ends the current iteration.

## Quiz

Answer the quiz for this chapter online at www.cs.armstrong.edu/liang/cpp3e/quiz.html.

## Programming Exercises

**Pedagogical Note**

Read each problem several times until you understand it. Think how to solve the problem before starting to write code. Translate your logic into a program. A problem often can be solved in many different ways. Students are encouraged to explore various solutions.

read and think before coding

explore solutions

### Sections 5.2–5.7

***5.1** (*Count positive and negative numbers and compute the average of numbers*) Write a program that reads an unspecified number of integers, determines how many positive and negative values have been read, and computes the total and average of the input values (not counting zeros). Your program ends with the input `0`. Display the average as a floating-point number. Here is a sample run:

```
Enter an integer, the input ends if it is 0: 1 2 -1 3 0 Enter
The number of positives is 3
The number of negatives is 1
The total is 5
The average is 1.25
```

```
Enter an integer, the input ends if it is 0: 0 Enter

No numbers are entered except 0
```

**5.2** (*Repeat multiplications*) Listing 5.4, `SubtractionQuizLoop.cpp`, generates five random subtraction questions. Revise the program to generate nine random multiplication questions for three integers between `1` and `5`. Display the correct count and test time.

**5.3** (*Conversion from millimeters to inches*) Write a program that displays the following table (note that `1` millimeter is `0.039` inches):

```
Millimeters     Inches
2               0.078
4               0.156
...
96              3.744
98              3.822
```

**5.4** (*Conversion from meters to feet*) Write a program that displays the following table (note that `1` meter is `3.280` feet):

```
Meters          Feet
1               3.280
2               6.560
...
14              45.920
15              49.200
```

**5.5** (*Conversion from millimeters to inches and inches to millimeters*) Write a program that displays the following tables side by side (note that `1` millimeter is `0.039` inches):

```
Millimeters     Inches    |   Inches      Millimeters
2               0.078     |   1           25.641
4               0.156     |   2           51.282
...
98              3.822     |   49          1256.410
100             3.900     |   50          1282.051
```

**5.6** (*Conversion from meters to feet*) Write a program that displays the following tables side by side (note that `1` meter is `3.280` feet):

```
Meters      Feet          |     Feet        Meters
1           3.280         |     3           0.915
2           6.560         |     6           1.829
...
14          45.920        |     42          12.805
15          49.200        |     45          13.720
```

**5.7** (*Use trigonometric functions*) Print the following table to display the `tan` and `cot` values of degrees from `0` to `60` with increments of `10` degrees. Round the value to keep four digits after the decimal point.